符号及定义：

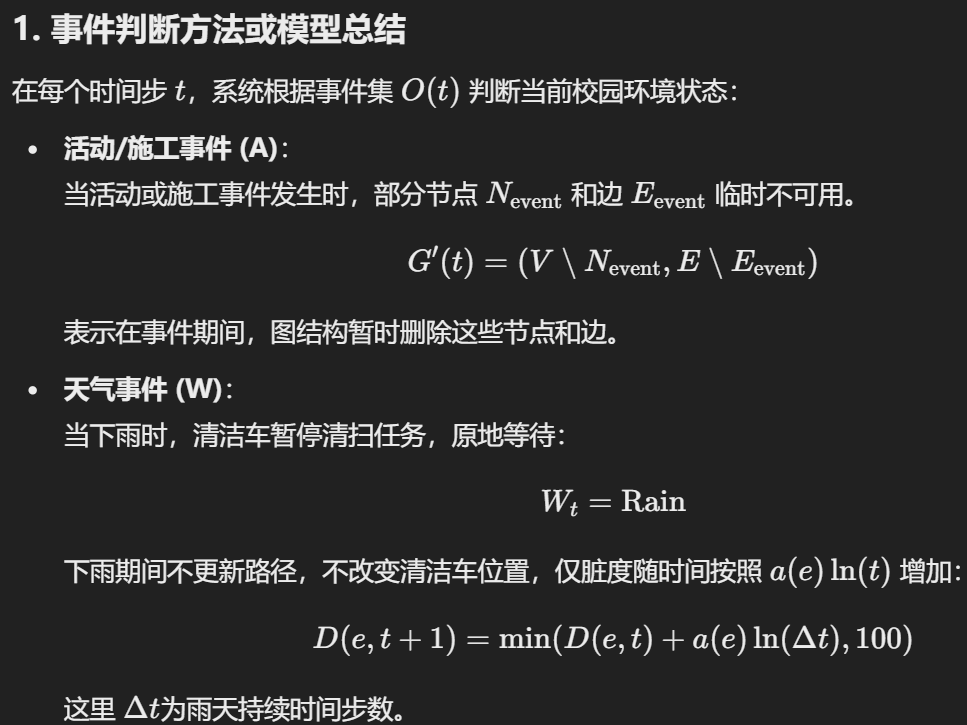


**模型建立过程：**

模型建立过程分为以下大步骤：

1. 事件判断与图结构更新
2. 无事件状态下的动态A\*路径规划
3. 事件发生时的动态路径规划调整
4. 事件结束后的脏度提升与车辆协同调配

**\*\*注意以下素材均以“截图”+“文字素材”+“伪代码”形式给出，方便查找\*\***



**文字素材：**

在每个时间步 𝑡，系统根据事件集 𝑂(t) 判断当前校园环境状态：

活动/施工事件 (A)：

当活动或施工事件发生时，部分节点和边临时不可用。

表示在事件期间，图结构暂时删除这些节点和边。

天气事件 (W)：

当下雨时，清洁车暂停清扫任务，原地等待：

下雨期间不更新路径，不改变清洁车位置，仅脏度随时间按照 𝑎(𝑒)ln(t)增加：

这里Δt为雨天持续时间步数。

**伪代码：**

**Function check\_events(t):**

**# Input: Time step t**

**# Output: Event status O(t)**

**events ← O(t) # Get the event status at time step t**

**If "Event" ∈ events or "Construction" ∈ events:**

**# Mark affected nodes and edges**

**N\_event ← affected\_nodes(events)**

**E\_event ← affected\_edges(events)**

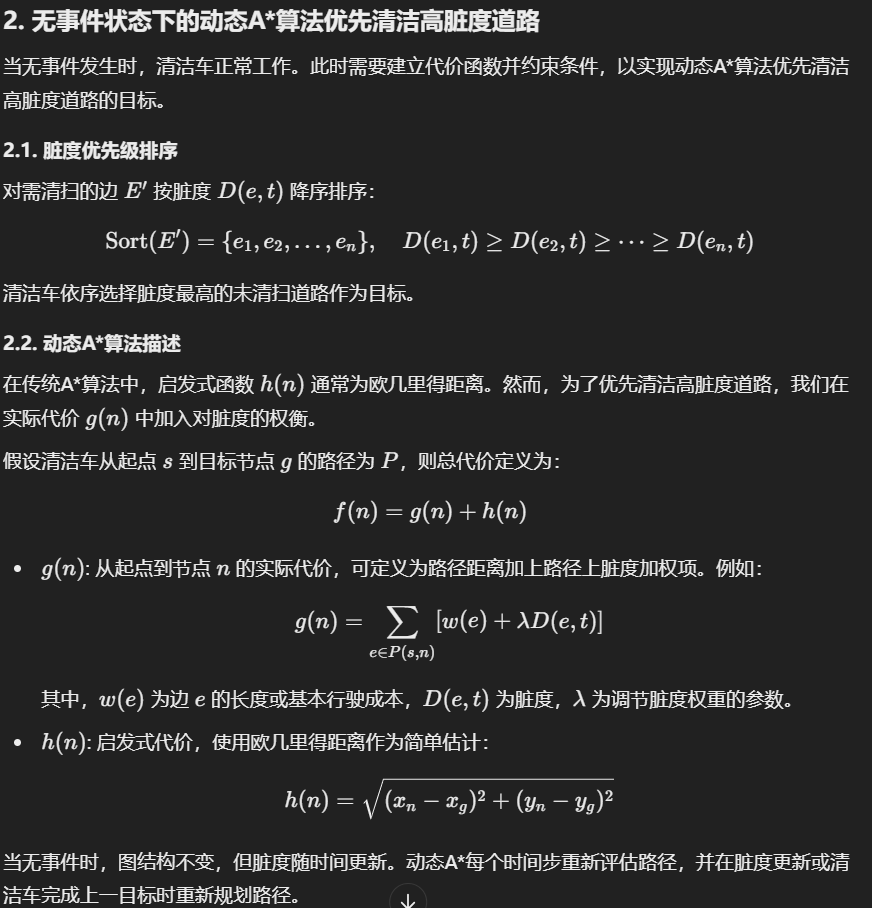
**# Update graph G'(t) = (V \ N\_event, E \ E\_event)**

**G' ← G(t) \ (N\_event, E\_event)**

**If "Rain" ∈ events:**

**pause\_cleaning()**

**Return events**



**文字素材：**

当无事件发生时，清洁车正常工作。此时需要建立代价函数并约束条件，以实现动态A\*算法优先清洁高脏度道路的目标。

2.1. 脏度优先级排序

对需清扫的边E'按脏度D(e,t)降序排序：

清洁车依序选择脏度最高的未清扫道路作为目标。

2.2. 动态A\*算法描述

在传统A\*算法中，启发式函数h(n）通常为欧几里得距离。然而，为了优先清洁高脏度道路，我们在实际代价g(n）中加入对脏度的权衡。

假设清洁车从起点s到目标节点g的路径为P，则总代价定义为：f(n)=g(n)+h(n)

g(n): 从起点到节点 n 的实际代价，定义为路径距离加上路径上脏度加权项。例如：

其中，w(e) 为边 e 的长度，D(e,t) 为脏度，λ为调节脏度权重的参数。

h(n): 启发式代价，使用欧几里得距离作为简单估计：

当无事件时，图结构不变，但脏度随时间更新。动态A\*每个时间步重新评估路径，但是为了避免清洁车路径过于频繁更换导致的清洁效率降低，我们给清洁车加上路径保护机制，只有清洁车完成上一目标时才重新规划路径。

**伪代码：**

**Function dynamic\_A\_star(graph, cleaner, target\_node, λ):**

**open\_list ← priority\_queue()**

**closed\_set ← set()**

**start ← cleaner.current\_node**

**g\_score, f\_score ← dict(), dict()**

**For v ∈ graph.V:**

**g\_score[v], f\_score[v] ← infinity, infinity**

**g\_score[start], f\_score[start] ← 0, h(start, target\_node)**

**open\_list.push((f\_score[start], start))**

**While open\_list not empty:**

**\_, current ← open\_list.pop()**

**If current = target\_node:**

**Return reconstruct\_path(came\_from, current)**

**closed\_set.add(current)**

**For neighbor ∈ graph.neighbors(current):**

**If neighbor ∈ closed\_set: Continue**

**tentative\_g ← g\_score[current] + w((current, neighbor)) + λ \* D((current, neighbor), t)**

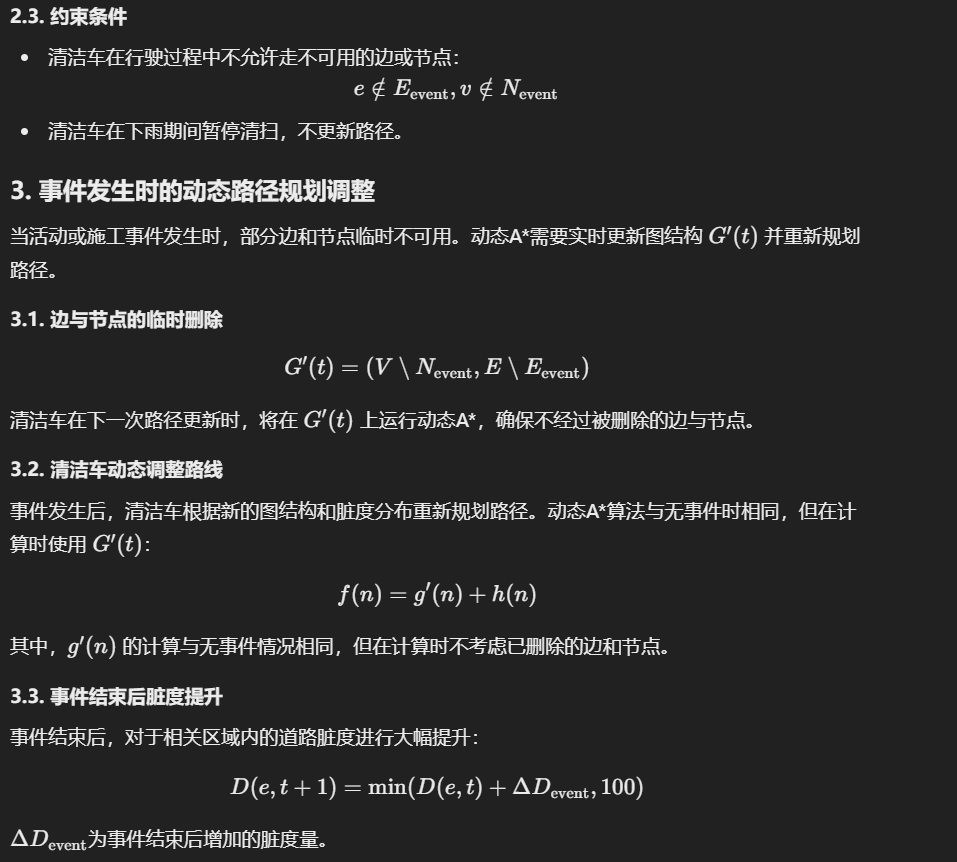
**If tentative\_g < g\_score[neighbor]:**

**came\_from[neighbor] ← current**

**g\_score[neighbor], f\_score[neighbor] ← tentative\_g, g\_score[neighbor] + h(neighbor, target\_node)**

**If neighbor not in open\_list:**

**open\_list.push((f\_score[neighbor], neighbor))**



**文字素材：**

* 清洁车在行驶过程中不允许走不可用的边或节点：
* 清洁车在下雨期间暂停清扫，不更新路径。

3. 事件发生时的动态路径规划调整

当活动或施工事件发生时，部分边和节点临时不可用。动态A\*需要实时更新图结构 G′(t) 并重新规划路径。

3.1. 边与节点的临时删除

清洁车在下一次路径更新时，将在 G′(t) 上运行动态A\*，确保不经过被删除的边与节点。

3.2. 清洁车动态调整路线

事件发生后，清洁车根据新的图结构和脏度分布重新规划路径。动态A\*算法与无事件时相同，但在计算时使用 G′(t)：f(n)=g′(n)+h(n)

其中，g′(n) 的计算与无事件情况相同，但在计算时不考虑已删除的边和节点。

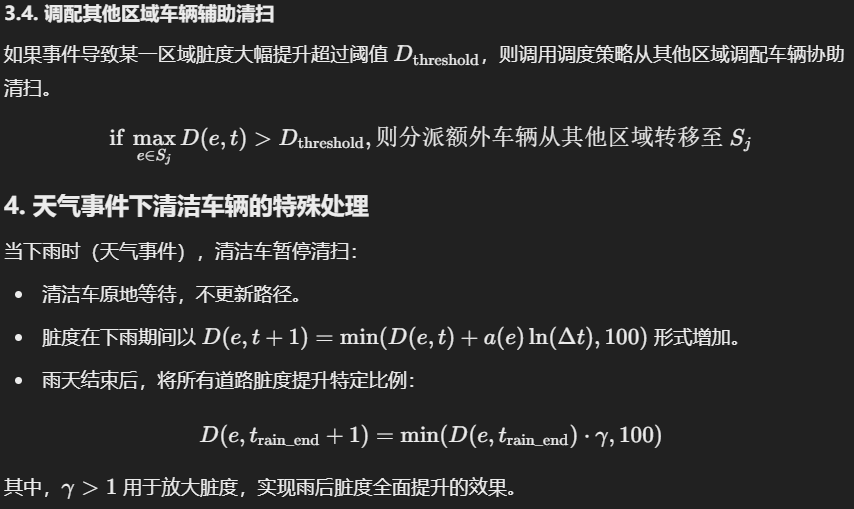
3.3. 事件结束后脏度提升

事件结束后，对于相关区域内的道路脏度进行大幅提升：

ΔDevent为事件结束后增加的脏度量。

3.4. 调配其他区域车辆辅助清扫

如果事件导致某一区域脏度大幅提升超过阈值 Dthreshold，则调用调度策略从其他区域调配车辆协助清扫。



**文字素材：**

则分派额外车辆从其他区域转移至 Sj

4. 天气事件下清洁车辆的特殊处理

当下雨时（天气事件），清洁车暂停清扫：

* 清洁车原地等待，不更新路径。
* 脏度在下雨期间以 形式增加。
* 雨天结束后，将所有道路脏度提升特定比例：

其中，γ>1用于放大脏度，实现雨后脏度全面提升的效果。

**综合伪代码流程：**

**Function main\_loop():**

**t ← 0**

**While system\_running:**

**events ← check\_events(t)**

**If "rain" ∈ events:**

**update\_dirtiness\_rain\_mode(G, a, t)**

**Else:**

**If "event" ∈ events or "construction" ∈ events:**

**G'(t) ← update\_graph\_for\_events(G, N\_event, E\_event)**

**Else:**

**G'(t) ← G**

**For cleaner ∈ C:**

**If cleaner.path is empty:**

**target\_edge ← select\_high\_dirtiness\_edge(G'(t), D)**

**λ ← choose\_lambda\_param()**

**path ← dynamic\_A\_star(G'(t), cleaner, target\_edge, λ)**

**cleaner.set\_path(path)**

**Else:**

**cleaner.move\_along\_path()**

**update\_dirtiness(G, cleaner)**

**If event\_ended:**

**restore\_graph(G'(t), G)**

**increase\_dirtiness\_post\_event(G, E\_event, deltaD\_event)**

**If need\_reallocate\_vehicles(G, D\_threshold):**

**reallocate\_vehicles(C, G, D, D\_threshold)**

**t ← t + 1**

**数学公式与模型细节总结：**

